BEGINNING RUBY

Lets learn ruby basics

INTRODUCTION TO RBENV

INSTALLATION

```
# Install rbenv
$ git clone https://github.com/rbenv/rbenv.git ~/.rbenv
$ echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bashrc
$ echo 'eval "$(rbenv init -)"' >> ~/.bashrc
$ exec $SHELL

# Install ruby-build
# ruby-build is an rbenv plugin that provides an rbenv install
#   command to compile and install different versions of Ruby
$ git clone https://github.com/rbenv/ruby-build.git
    ~/.rbenv/plugins/ruby-build
$ echo 'export PATH="$HOME/.rbenv/plugins/ruby-build/bin:$PATH"'
    >> ~/.bashrc
$ exec $SHELL

# Installing rbenv
```

USAGE:

Installing Ruby versions

```
# list all available versions:
rbenv install -l

# install a Ruby version:
rbenv install 2.4.1

# Uninstall Ruby 2.1.0
rbenv uninstall 2.1.0
```

RUBY BASICS

LETS BEGIN OUR JOURNEY ON RUBY WITH IRB

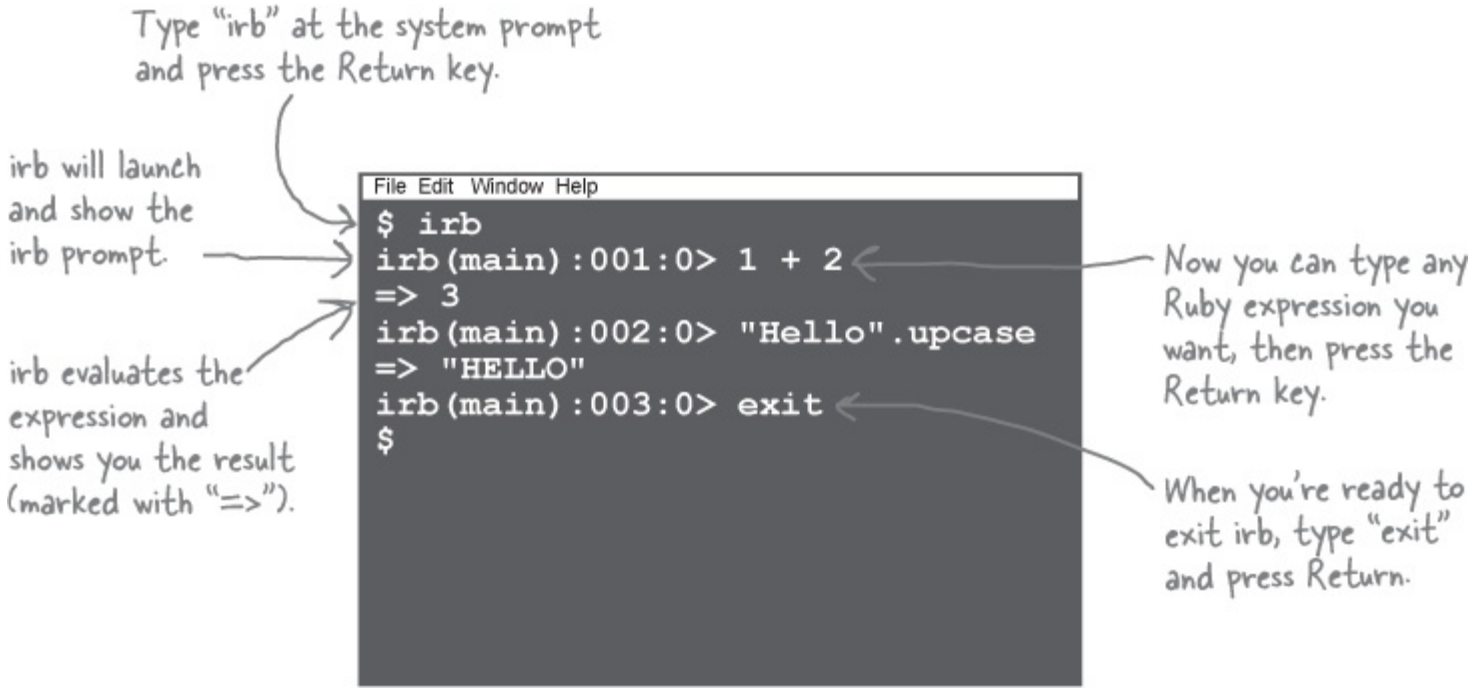
IRB

Interactive Ruby Shell (IRB or irb) is a REPL for programming in the object-oriented scripting language Ruby. The abbreviation irb comes from the fact that the filename extension for Ruby is ".rb", although interactive Ruby files do not have an extension of ".irb".

"REPL": Read, Evaluate, Print, Loop.

Think of it kind of like a calculator. Each complete block of codes get executed and the result is displayed just below the code

IRB SCREENSHOT



LETS LEARN RUBY WITH IRB

VARIABLES:

- Named location in memory that can store data.
- Flexible - can hold any kind of data
- No declaration required
- Store valid datatypes

EXAMPLE:

```
irb(main):001:0> a=5
=> 5
irb(main):002:0> b="hello"
=> "hello"
irb(main):003:0> a=b
=> "hello"
irb(main):004:0> a=5
=> 5
irb(main):005:0> b="hello"
=> "hello"
irb(main):016:0> c = a + 4
=> 9
irb(main):007:0> a=b
=> "hello"
```

NAMING VARIABLES

Most Ruby variables (local variables) have a few requirements imposed by the VM. They…

- always start with a lowercase letter (underscore is permitted, though uncommon)
- have no spaces
- do not contain most special characters like $, @, and &

RUBY VARIABLE NAMING CONVENTIONS

- use snake case where each word in the name is lowercase and connected by underscores (_)
- are named after the meaning of their contents, not the type of their contents
- aren’t abbreviated

  Good variable names might be count, students_in_class, or first_lesson.

RUBY VARIABLE BAD NAMING EXAMPLES

A few examples of bad Ruby variable names include:

- studentsInClass – uses camel-case rather than snake-case, should be students%_in%_class
- 1st_lesson – variables can’t start with a number, should just be first_lesson
- students_array – includes the type of the data in the name, should just be students
- sts – abbreviates rather than just using students

ARE THESE VALID VARIABLE NAMES ?

- time_machine
- student_count_integer
- homeworkAssignment
- 3_sections
- top_ppl

STRINGS

- collections of letters and numbers
- in between single quotes (') or double quotes (")
- could be a single letter like "a", a word like "hi", or a sentence like "Hello my friends.".
- empty string: "" [String non the less]
- Double-quoted strings allow substitution and backslash notation but single-quoted strings don't allow substitution and allow backslash notation only for \\ and \'
- substitute the value of any Ruby expression into a string using the sequence #{ expr }

STRING EXAMPLES

```
irb(main):023:0> 'escape using "\\"'
=> "escape using \"\\\""
irb(main):024:0> 'That\'s right'
=> "That's right"
irb(main):025:0> "escape using '\\'"
=> "escape using '\\'"
irb(main):026:0> "escape using \\"
=> "escape using \\"
irb(main):028:0> puts "escape using '\\'"
escape using '\'
=> nil
irb(main):037:0> "Multiplication Value : #{24*60*60}"
=> "Multiplication Value : 86400"
```

SUBSTRINGS

- a part of the whole string

```
irb(main):039:0> greeting = "Hello Everyone!"
=> "Hello Everyone!"
irb(main):040:0> greeting[0..4]
=> "Hello"
irb(main):041:0> greeting[6..14]
=> "Everyone!"
irb(main):042:0> greeting[6..-1]
=> "Everyone!"
irb(main):043:0> greeting[6..-2]
=> "Everyone"
```

POSITIVE AND NEGATIVE POSITIONS

- The characters in a string each have a position number, starting with zero. So for a string "Hi", the "H" is in position zero and the "i" is in position 1.
- To pull out a substring we use the starting and ending positions. Thus greeting[0..4] above pull out the letters in position zero, one, two, three, and four.
- Ruby interprets negative positions to count back from the end of the string. So in "Hi", the "i" is in position -1 and the "H" is in position -2.

COMMON STRING METHODS

.LENGTH

```
irb(main):044:0> greeting = "Hello Everyone!"
=> "Hello Everyone!"
irb(main):045:0> greeting.length
=> 15
```

.SPLIT

```
irb(main):053:0> sentence = "This is my sample sentence."
=> "This is my sample sentence."
irb(main):054:0> sentence.split
=> ["This", "is", "my", "sample", "sentence."]
irb(main):055:0> numbers = "one,two,three,four,five"
=> "one,two,three,four,five"
irb(main):056:0> numbers.split
=> ["one,two,three,four,five"]
irb(main):057:0> numbers.split(",")
=> ["one", "two", "three", "four", "five"]
```

.SUB AND .GSUB

- Substutions

```
irb(main):068:0> comment = "I really really like the real Ruby"
=> "I really really like the real Ruby"
irb(main):069:0> comment.sub("real", "not_real")
=> "I not_really really like the real Ruby"
irb(main):070:0> comment.gsub("real", "not_real")
=> "I not_really not_really like the not_real Ruby"
```

STRING CONCATENATION

```
irb(main):072:0> name = "Ed"
=> "Ed"
irb(main):073:0> puts "Good morning, " + name + "!"
Good morning, Ed!
=> nil
```

STRING INTERPOLATION

```
irb(main):074:0> name = "Ed"
=> "Ed"
irb(main):075:0> puts "Good morning, #{name}!"
Good morning, Ed!
=> nil
```

.NIL?

```
irb(main):001:0> comment = "I really really like the real Ruby"
=> "I really really like the real Ruby"
irb(main):002:0> comment.nil?
=> false
irb(main):003:0> comment = ""
=> ""
irb(main):004:0> comment.nil?
=> false
irb(main):005:0> comment = nil
=> nil
irb(main):006:0> comment.nil?
=> true
```

SYMBOLS

- starts with a colon then one or more letters, like :flag or :best_friend.
- look like strings but behave like numbers

STRINGS VS SYMBOLS

- **Symbols in Ruby are basically "immutable strings"** .. that means that they can not be changed, and it implies that the same symbol when referenced many times throughout your source code, is always stored as the same entity, e.g. has the same object id.
- **Strings on the other hand are mutable**
- Using symbols not only saves time when doing comparisons, but also saves memory, because they are only stored once.

think of a symbol as a "named integer".

TRY THESE

```
irb(main):088:0> "hello".methods
irb(main):089:0> "hello".methods.count
irb(main):090:0> :hello.methods
irb(main):091:0> :hello.methods.count
```

NUMBERS

- two basic kinds: integers (whole numbers) and floats (have a decimal point).

```
irb(main):092:0> 10/2
=> 5
irb(main):093:0> 11/2
=> 5
irb(main):094:0> 2*5
=> 10
irb(main):095:0> 11/2.0
=> 5.5
irb(main):096:0> 5*2
=> 10
irb(main):098:0> 2.3*5
=> 11.5
irb(main):099:0> 2.0*5
=> 10.0
irb(main):100:0> 2+5.0
=> 7.0
```

SEQUENTIAL LOOPS

other languages -> for .. loop

```
for(var i = 0; i < 5; i++){
    console.log("Hello, World");
}
```

- For loops are common, but they’re not very readable.
- ruby sequential loops are readable.

```
irb(main):104:0> 5.times do
irb(main):105:1*   puts "Hello, World!"
irb(main):106:1> end
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
=> 5
```

BLOCKS

1. do .. end style
2. Brack Blocks { .. }

```
irb(main):104:0> 5.times do
irb(main):105:1*   puts "Hello, World!"
irb(main):106:1> end
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
=> 5
irb(main):107:0> 5.times{ puts "Hello, World!" }
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
=> 5
```

BLOCKS

are parameter passed into a method call

- If, for instance, we just called 5.times, Ruby wouldn’t know what we want to be done five times. When we pass in the block we’re saying "here are the instructions I want you to run each time".
- There are many methods that accept blocks. Like the .gsub method you saw on String earlier will run a block once for each match:

```
irb(main):108:0> "this is a sentence".gsub("e"){ puts "Found an E!"}
Found an E!
Found an E!
Found an E!
=> "this is a sntnc"
```

BLOCK PARAMETERS

```
irb(main):112:0> 5.times do |i|
irb(main):113:1*   puts "#{i}: Hello, World!"
irb(main):114:1> end
0: Hello, World!
1: Hello, World!
2: Hello, World!
3: Hello, World!
4: Hello, World!
=> 5
irb(main):115:0> "this is a sentence".gsub("e"){|letter| letter.upca
=> "this is a sEntEncE"
```

ARRAYS

```
irb(main):116:0> meals = ["Breakfast", "Lunch", "Dinner"]
=> ["Breakfast", "Lunch", "Dinner"]
irb(main):117:0> meals < "Dessert"
=> ["Breakfast", "Lunch", "Dinner", "Dessert"]
irb(main):118:0> meals[2]
=> "Dinner"
irb(main):119:0> meals[0]
=> "Breakfast"
irb(main):120:0> meals.first
=> "Breakfast"
irb(main):121:0> meals[-1]
=> "Dessert"
irb(main):122:0> meals.last
=> "Dessert"
irb(main):123:0> meals[2..3]
=> ["Dinner", "Dessert"]
```

COMMON ARRAY METHODS

```
irb(main):139:0>     a = [ "d", "a", "e", "c", "b" ]
=> ["d", "a", "e", "c", "b"]
irb(main):140:0>     a.sort
=> ["a", "b", "c", "d", "e"]
irb(main):141:0>     a.sort { |x,y| y <=> x }
=> ["e", "d", "c", "b", "a"]
irb(main):142:0>     [3, 5.5].sum
=> 8.5
irb(main):143:0>     ["a", "b", "c"].join
=> "abc"
irb(main):144:0>     ["This", "is", "my", "sample", "sentence."].joi
=> "This is my sample sentence."
```

ARRAY LOOPING

```
irb(main):162:0> arr = [ 11, 12, 13]
=> [11, 12, 13]
irb(main):163:0> arr.each { |a| print a -= 10, " " }
1 2 3 => [11, 12, 13]
irb(main):164:0> arr.each_with_index {|a,i| puts "#{i}: #{a}" }
0: 11
1: 12
2: 13
=> [11, 12, 13]
```

More on Array

HASH

- Key/Value Pairs
- *unordered* collection

```
irb(main):170:0> produce = {"apples" => 3, "oranges" => 1, "carrots"
=> {"apples"=>3, "oranges"=>1, "carrots"=>12}
irb(main):171:0> puts "There are #{produce['oranges']} oranges in th
There are 1 oranges in the fridge.
=> nil
irb(main):172:0> produce["grapes"] = 221
=> 221
irb(main):173:0> produce
=> {"apples"=>3, "oranges"=>1, "carrots"=>12, "grapes"=>221}
irb(main):174:0> produce["oranges"] = 6
=> 6
irb(main):175:0> produce
=> {"apples"=>3, "oranges"=>6, "carrots"=>12, "grapes"=>221}
irb(main):176:0> produce.keys
=> ["apples", "oranges", "carrots", "grapes"]
irb(main):177:0> produce.values
```

COMMON HASH METHODS

```
irb(main):183:0> h = { "a" => 100, "b" => 200 }
=> {"a"=>100, "b"=>200}
irb(main):184:0> h.each {|key, value| puts "#{key} is #{value}" }
a is 100
b is 200
=> {"a"=>100, "b"=>200}
irb(main):185:0> h.has_key?("a")   #=> true
=> true
irb(main):186:0> h.value?(100)   #=> true
=> true
irb(main):187:0> h.value?(999)   #=> false
=> false
irb(main):188:0> h.key(200)
=> "b"
irb(main):189:0> h.length
=> 2
```

More on Hash

RUNNING RUBY FILE FROM TERMINAL/COMMANDLINE

HELLO WORLD

- Write this in your file `hello_world.rb`

```
puts "Hello, World!"
```

- In the terminal go to files path, and run the file

```
cd /path/to/file
ruby hello_world.rb
```

LOAD RUBY FILE TO IRB

```
irb(main):001:0> load 'hello_world.rb'
Hello, World!
=> true
```

LEARNING GIT

- Create ruby_hello_world folder
- Move hello_world.rb to ruby_hello_world
- git init
- git commit -m 'write hello world in ruby'

```
# create ruby_hello_world directory
mkdir ruby_hello_world

# move hello_world.rb to ruby_hello_world
mv hello_world.rb ruby_hello_world/

# initialize a git repository
git init

# add hello_world.rb to be tracked
git add hello_world.rb

# make a commit to track the current state of file
git commit -m 'write hello world in ruby'
```

CONDITIONALS

- evaluates to `true` or `false`
- operators :
  - == (equal)
  - > (greater than)
  - >= (greater than or equal to)
  - < (less than)
  - <= (less than or equal to).

CONDITIONALS EXAMPLES

```
irb(main):001:0> true
=> true
irb(main):002:0> false
=> false
irb(main):003:0> 1 == 2
=> false
irb(main):004:0> b = 4
=> 4
irb(main):005:0> b < 2
=> false
irb(main):007:0> b.nil?
=> false
irb(main):009:0> b >= 2
=> true
```

CONDITIONAL BRANCHING / INSTRUCTIONS

- `if` / `elsif` / `else` structures

put the followin in water_status.rb

```
def water_status(minutes)
if minutes < 7
    puts "The water is not boiling yet."
  elsif minutes == 7
    puts "It's just barely boiling"
  elsif minutes == 8
    puts "It's boiling!"
  else
    puts "Hot! Hot! Hot!"
  end
end
```

CONDITIONAL BRANCHING / INSTRUCTIONS

In the irb run

```
irb(main):001:0> load 'water_status.rb'
=> true
irb(main):002:0> water_status(6)
The water is not boiling yet.
=> nil
irb(main):003:0> water_status(7)
It's just barely boiling
=> nil
```

CREATING CLASSES IN RUBY

- Creating a singer class `student.rb`

```ruby
# singer.rb
class Singer

end
```

- Inside the class we usually define one or more methods using the def keyword:

```ruby
# singer.rb
class Singer
  def introduction
    puts "Hi, I'm #{first_name}!"
  end
end
```

LETS ADD FIRST_NAME, :LAST_NAME, AND :PRIMARY_PHONE_NUMBER

```ruby
# singer.rb
class Singer
  attr_accessor :first_name, :last_name, :primary_phone_number

  def introduction
    puts "Hi, I'm #{first_name}!"
  end
end
```

CREATE AN INSTANCE

```ruby
# singer.rb
class Singer
  attr_accessor :first_name, :last_name, :fav_songs

  def introduction
    puts "Hi, I'm fan of #{first_name} #{last_name}!"
  end
end

ed = Singer.new
ed.first_name = "Ed"
ed.last_name = "Sheeran"
ed.introduction
```

MY FAV_SONGS

```
ed.fav_songs = ['Nancy Mulligan', 'Shape of You', 'Thinking Out Loud
```

```ruby
# singer.rb
class Singer
  attr_accessor :first_name, :last_name, :fav_songs

  def introduction
    puts "Hi, I'm fan of #{first_name} #{last_name}!"
  end

  def list_fav_songs
    puts "I feel like eloping after listening to #{fav_songs[0]}"
    puts "Can't stop singing #{fav_songs[1]}"
    puts "Really love #{fav_songs[-1]}"
  end
end

ed = Singer.new
```

ASSIGNMENT

1. **Create an account in**

   GITHUB.COM

2. **In GITHUB.COM, create a project named in**

   ruby_hello_world

REFERENCE

http://tutorials.jumpstartlab.com/projects/ruby_in_100_minute